2

AVF Control Number: AVF-VSR-90502/54

Ada COMPILER
VALIDATION SUMMARY REPORT:

Certificate Number: #890920N1.10170
SD-Scicon plc
XD Ada MIL-STD-1750A T1.0-05A
VAX Cluster Host and Fairchild F9450 on a SBC-50 board (MIL-STD-1750A) (Bare machine)

Completion of On-Site Testing:
20 September 1989

Prepared By:
Testing Services
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

OFFICE OF THE DIRECTOR OF
DEFENSE RESEARCH AND ENGINEERING

WASHINGTON, DC 20301

19 APR 1990

(R&AT)

MEMORANDUM FOR Director, Directorate of Database Services,
Defense Logistics Agency

SUBJECT: Technology Screening of Unclassified/Unlimited Reports

Feb 12
per
Ann
Easter

Your letter of 2 February 1990 to the Commander, Air Force Systems Command, Air Force Aeronautical Laboratory, Wright-Patterson Air Force Base stated that the Ada Validation Summary report for Meridian Software Systems, Inc. contained technical data that should be denied public disclosure according to DoD Directive 5230.25

We do not agree with this opinion that the contents of this particular Ada Validation Summary Report or the contents of the several hundred of such reports produced each year to document the conformity testing results of Ada compilers. Ada is not used exclusively for military applications. The language is an ANSI Military Standard, a Federal Information Processing Standard, and an International Standards Organization standard. Compilers are tested for conformity to the standard as the basis for obtaining an Ada Joint Program Office certificate of conformity. The results of this testing are documented in a standard form in all Ada Validation Summary Reports which the compiler vendor agrees to make public as part of his contract with the testing facility.

On 18 December 1985, the Commerce Department issued Part 379 Technical Data of the Export Administration specifically listing Ada Programming Support Environments (including compilers) as items controlled by the Commerce Department. The AJPO complies with Department of Commerce export control regulations. When Defense Technical Information Center receives an Ada Validation Summary Report, which may be produced by any of the five U.S. and European Ada Validation Facilities, the content should be made available to the public.

If you have any further questions, please feel free to contact the undersigned at (202) 694-0209.

John P. Solomond

John P. Solomond
Director
Ada Joint Program Office

URITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: SD-Scicon XD Ada MIL-STD-1750A T1.0-05A, VAX Cluster (Host) to Fairchild F9450 on a SBC-50 board (Target), 890920N1.10170		5. TYPE OF REPORT & PERIOD COVERED 20 Sept. 1989 to 20 Sept. 1990 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) National Computing Centre Limited, Manchester, United Kingdom.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS National Computing Centre Limited, Manchester, United Kingdom.		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		12. REPORT DATE 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) National Computing Centre Limited, Manchester, United Kingdom.		15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 If different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SD-Scicon plc, XD Ada MIL-STD-1750A T1.0-05A, Manchester, United Kingdom, VAX Cluster (Comprising of a VAX 8600 and 7 MicroVAX II's) to Fairchild F9450 on a SBC-50 board (MIL-STD-1750A)(bare machine)(Target), ACVC 1.10

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Enter

Ada Compiler Validation Summary Report:

Compiler Name: XD Ada MIL-STD-1750A T1.0-05A

Certificate Number: #890920N1.10170

Host: VAX Cluster (Comprising of a VAX 8600 and 7 MicroVAX II's)

Target: Fairchild F9450 on a SBC-50 board (MIL-STD-1750A) (bare machine)

Testing Completed 20 September 1989 Using ACVC 1.10

This report has been reviewed and is approved.

Jane Pink
Testing Services Manager
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director AJPO
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies -- for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by The National Computer Centre Limited according to procedures established by the Ada Joint Program Office and administered by the Ada Validation

Organization (AVO). On-site testing was completed 20 September 1989 at SD-SCICON plc, Pembroke House, Pembroke Broadway, Camberley, Surrey, GU15 3XD, UK.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Testing Services
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC
The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.

Ada Commentary
An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.

Ada Standard
ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

Applicant
The agency requesting validation.

AVF
The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.

AVO
The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.

Compiler
A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test
An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host
The computer on which the compiler resides.

Inapplicable test
An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test
An ACVC test for which a compiler generates the expected result.

Target — The computer which executes the code generated by the compiler.

Test — A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn test — An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters -- for example, the number of identifiers permitted in a compilation or the number of units in a library -- a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time -- that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values -- for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler:	XD Ada MIL-STD-1750A T1.0-05A	
ACVC Version:	1.10	
Certificate Number:	#890920N1.10170	
Host Computer:		
Machine:	VAX Cluster (comprising of a VAX 8600 and 7 MicroVAX II's)	
Operating System:	VMS 5.1	
Memory Size:	VAX 8600 - MicroVAX II's -	20Mbytes 1 x 16 Mbytes 6 x 9 Mbytes
Target Computer:		
Machine:	Fairchild F9450 on a SBC-50 board (MIL-STD-1750A) (bare machine)	
Memory Size:	1 Mb	
Communications Network:	RS232 link	

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

(1) This implementation supports the additional predefined types LONG_INTEGER and LONG_FLOAT, in the package STANDARD. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B).

(3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)

(4) NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is not gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is inconsistant. (See tests C46012A..Z (26 tests).)

(2) The method used for rounding to longest integer is round away from zero. See tests C46012A..Z (26 tests).)

(3) The method used for rounding to integer in static universal real expressions is round to odd. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR. (See test C36003A.)

(2) CONSTRAINT_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

(3) CONSTRAINT_ERROR is raised when an array type with SYSTEM.MAX_INT + 2 components is declared. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises no exception. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

(6) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(7) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no exception. (See test E52103Y.)

g. Discriminated types.

(1) In assigning record types with discriminants, the exprcssion is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

h. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

i. Pragmas.

(1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

j. Generics.

(1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

(2) Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)

(3) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)

(4) Generic non-library package bodies as subunits can be compiled in separate compilations. (See test CA2009C.)

(5) Generic non-library subprogram bodies can be compiled in separate compilations from their stubs. (See test CA2009F.)

(6) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

(7) Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

(8) Generic library package specifications and bodies can be compiled in separate compilations. (See tests BC3204C and BC3205D.)

(9) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

k. Input and output.

(1) The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

(2) The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

(3) The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO.

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 641 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 285 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 11 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	129	1131	1693	17	16	46	3032
Inapplicable	0	7	622	0	12	0	641
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	192	547	496	245	172	99	160	331	137	36	252	287	78	3032
Inapp	20	102	184	3	0	0	6	1	0	0	0	82	243	641
Withdrawn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

E28005C A39005G B97102E
C97116A BC3009B CD2A62D
CD2A63A..D (4 tests) CD2A66A..D (4 tests) CD2A73A..D (4 tests)
CD2A76A..D (4 tests) CD2A81G CD2A83G
CD2A84M..N (2 tests) CD2B15C CD2D11B
CD5007B CD5011O ED7004B
ED7005C..D (2 tests) ED7006C..D (2 tests) CD7105A
CD7203B CD7204B CD7205C
CD7205D CE2107I CE3111C
CE3301A CE3411B

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 641 tests were inapplicable for the reasons indicated:

a. The following 285 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113F..Y (20 tests) C35705F..Y (20 tests) C35706F..Y (20 tests)
C35707F..Y (20 tests) C35708F..Y (20 tests) C35802F..Z (21 tests)
C45241F..Y (20 tests) C45321F..Y (20 tests) C45421F..Y (20 tests)
C45521F..Z (21 tests) C45524F..Z (21 tests) C45621F..Z (21 tests)
C45641F..Y (20 tests) C46012F..Z (21 tests)

b. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

c. The following 16 tests are not applicable because this implementation does not support a predefined type SHORT_INTEGER:

C45231B C45304B C45502B C45503B C45504B
C45504E C45611B C45613B C45614B C45631B
C45632B B52004E C55B07B B55B09D B86001V
CD7101E

d. C45531M..P (4 tests) and C45532M..P (4 tests) are all inapplicable because this implementation has a 'MAX_MANTISSA of 31 and these tests require the compiler to support a greater value.

e. C86001F is not applicable because, for this implementation, the package TEXT_IO is dependent upon package SYSTEM. This test recompiles package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete.

f. B86001X, C45231D, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

g. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

h. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

i. C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

j. CD1009C, CD2A41A..B (2 tests), CD2A41E and CD2A42A..J (10 tests) are not applicable because 'SIZE representation clauses for floating-point types are not supported.

k. CD1C04C is inapplicable because this implementation does not support model numbers of a derived type that are not representable values of the parent type.

l. CD2A52C..D (2 tests), CD2A52G..H (2 tests), CD2A54C..D (2 tests) and CD2A54H are not applicable because for this implementation the legality of a 'SIZE clause for a derived fixed point type can depend on the representation chosen for the parent type.

m. CD2A52J and CD2A54J is not applicable because this tests require an unsigned representation for a fixed point type; this implementation does not support unsigned fixed point representation.

n. CD2A53C, and CD2A54G are not applicable because within these tests the SMALL specified for a derived fixed point is finer than the SMALL for the parent type. As a result some model numbers of the derived type are not representable values of the parent type which this implementation does not allow.

o. The following 23 tests are not applicable because this implementation does not support packing by means of a length clause for an array type:

CD2A61A..L (12 tests)	CD2A62A..C (3 tests)	CD2A64A..D (4 tests)

CD2A65A..D (4 tests)

p. The following 16 tests are not applicable because this implementation does not support packing by means of a length clause for a record type:

CD2A71A..D (4 tests) CD2A72A..D (4 tests) CD2A74A..D (4 tests)
CD2A75A..D (4 tests)

q. CD2A84B..I (8 tests) and CD2A84K..L (2 tests) are not applicable because this implementation only accepts length clause for access types, if the default size (32 bits) is specified. These tests specify sizes other that 32 bits.

r. CD2A91A..E (5 tests) are not applicable because this implementation does not support SIZE representation clauses for task types.

s. The following 241 tests are inapplicable because sequential, text, and direct access files are not supported:

CE2102A..C (3 tests) CE2102G..H (2 tests) CE2102K
CE2102N..Y (12 tests) CE2103C..D (2 tests) CE2104A..D (4 tests)
CE2105A..B (2 tests) CE2106A..B (2 tests) CE2107A..H (8 tests)
CE2107L CE2108A..H (8 tests) CE2109A..C (3 tests)
CE2110A..D (4 tests) CE2111A..I (9 tests) CE2115A..B (2 tests)
CE2201A..C (3 tests) EE2201D..E (2 tests) CE2201F..N (9 tests)
CE2204A..D (4 tests) CE2205A CE2208B
CE2401A..C (3 tests) EE2401D CE2401E..F (2 tests)
EE2401G CE2401H..L (5 tests) CE2404A..B (2 tests)
CE2405B CE2406A CE2407A..B (2 tests)
CE2408A..B (2 tests) CE2409A..B (2 tests) CE2410A..B (2 tests)
CE2411A CE3102A..B (2 tests) EE3102C
CE3102F..H (3 tests) CE3102J..K (2 tests) CE3103A
CE3104A..C (3 tests) CE3107B CE3108A..B (2 tests)
CE3109A CE3110A CE3111A..B (2 tests)
CE3111D..E (2 tests) CE3112A..D (4 tests) CE3114A..B (2 tests)
CE3115A EE3203A CE3208A
EE3301B CE3302A CE3305A
CE3402A EE3402B CE3402C..D (2 tests)
CE3403A..C (3 tests) CE3403E..F (2 tests) CE3404B..D (3 tests)
CE3405A EE3405B CE3405C..D (2 tests)
CE3406A..D (4 tests) CE3407A..C (3 tests) CE3408A..C (3 tests)
CE3409A CE3409C..E (3 tests) EE3409F
CE3410A CE3410C..E (3 tests) EE3410F
CE3411A CE3411C CE3412A
CE3413A CE3413C CE3602A..D (4 tests)
CE3603A CE3604A..B (2 tests) CE3605A..E (5 tests)
CE3606A..B (2 tests) CE3704A..F (6 tests) CE3704M..O (3 tests)
CE3706D CE3706F..G (2 tests) CE3804A..P (16 tests)

CE3805A..B (2 tests) CE3806A..B (2 tests) CE3806D..E (2 tests)
CE3806G..H (2 tests) CE3905A..C (3 tests) CE3905L
CE3906A..C (3 tests) CE3906E..F (2 tests)

t. CE3901A is not applicable because this implementation raises NAME_ERROR if a filename parameter to TEXT_IO.CREATE is non-null. This test assumes that USE_ERROR will be raised.

u. EE3412C is not applicable for this implementation because their implementation of the body of the package report does not use TEXT_IO.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behaviour. Modifications are made by the AVF in cases where legitimate implementation behaviour prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behaviour that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 11 tests.

C34006D is classified as passed if the test fails with messages "INCORRECT TYPE'SIZE" or "INCORRECT OBJECT'SIZE". This test assumes that the space allocated for objects must be less than or equal to the minimum needed by the (sub) type. This is not true for this implementation.

C45524A..E (5 tests) were modified because these tests expect that the result of continued division of a real number will be zero; the Ada Standard, however, only requires that the result be within the type's SAFE_SMALL of zero. Thus, these tests were modified to include a check that the result was in the smallest positive safe interval for the type. The implementation passed the modified tests. Each test was modified by inserting the following code after line 138;

```
ELSIF VAL <= F'SAFE_SMALL THEN
      COMMENT ("UNDERFLOW IS GRADUAL")
```

C64103A and C95084A were classified as passed although the following messages were output

```
*C64103A    EXCEPTION NOT RAISED AFTER CALL - P2(B)
*C95084A    EXCEPTION NOT RAISED AFTER CALL - T2(A)
*C95084A    EXCEPTION NOT RAISED AFTER CALL - T2(B)
```

This is accepted because for this implementation the range of FLOAT and LONG_FLOAT is the same and it is only the accuracy of the types that is different.

C64201C contains 12 tasks and at execution time the memory required for these exceeds that available for task activation on the target computer - STORAGE_ERROR IS RAISED. A modified version of the test using representation clauses to decrease the task size to 2K bytes was prepared and executed successfully. The compiler will also allow the default task size to be altered using a compiler option. This facility was tested and resulted in a test which executed successfully.

AD7006A is graded as passed as it compiles without error. The test attempts to convert SYSTEM.MEMORY_SIZE to type INTEGER. This result is accepted by the AVO.

The following test was split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B97103E

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the **XD Ada MIL-STD-1750A T1.0-05A** compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behaviour on all inapplicable tests.

3.7.2 Test Method

Testing of the **XD Ada MIL-STD-1750A T1.0-05A** compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware ~~and software~~ components:

Host computer	: **VAX Cluster (comprising of a VAX 8600 and 7 MicroVAX II's)**
Host operating system	: **VMS 5.1**
Target computer	: **Fairchild F9450 on a SBC-50 board (MIL-STD-1750A) (bare machine)**
Compiler	: **XD Ada MIL-STD-1750A T1.0-05**
Pre-linker	: **XD Ada MIL-STD-1750A T1.0-05**
Assembler	: **XD Ada MIL-STD-1750A T1.0-05**
Linker	: **XD Ada MIL-STD-1750A T1.0-05**
Loader/Downloader	: **XD Ada MIL-STD-1750A T1.0-05**
Runtime System	: **XD Ada MIL-STD-1750A T1.0-01**

The host and target computers were linked via a RS232 link.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were not included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the VAX Cluster, then all executable images were transferred to the MIL-STD-1750A target via the RS232 link and run. Results were printed from the host computer.

The compiler was tested using command scripts provided by SD-Scicon plc and reviewed by the validation team. The compiler was tested using all the following option settings. Details of these settings are given at the end of Appendix B.

Tests were compiled, linked, and executed (as appropriate) using 8 computers and a single target computer. Test output, compilation listings, and job logs were captured on magnetic media and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at SD-Scicon plc, Pembroke House, Pembroke Broadway, Camberley, Surrey, GU15 3XD, UK and was completed on 20 September 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

SD-Scicon plc has submitted the following Declaration of Conformance concerning the **XD Ada MIL-STD-1750A T1.0-05A** compiler.

DECLARATION OF CONFORMANCE

Compiler Implementor: **SD-Scicon plc**

Ada Validation Facility: **The National Computing Centre Limited**
Oxford Road
Manchester
M1 7ED

Ada Compiler Validation Capability (ACVC) Version: **1.10**

Base Configuration

Base Compiler Name: **XD Ada MIL-STD-1750A T1.0-05A**

Host Architecture: **VAX Cluster (comprising of a VAX 8600 and 7 MicroVAX II's)**

Host OS and Version: **VMS 5.1**

Target Architecture: **Fairchild F9450 on a SBC-50 board (MIL-STD-1750A) (bare machine)**

Implementor's Declaration

I, the undersigned, representing **SD-Scicon plc**, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that **SD-Scicon plc** is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

Date : 10th October 1989

Bill Davison
WORLDWIDE CUSTOMER SERVICES MANAGER

Owner's Declaration

I, the undersigned, representing **SD-Scicon plc**, take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/target performance, are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

Date : 10th October 1989

Bill Davison
WORLDWIDE CUSTOMER SERVICES MANAGER

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the **XD Ada MIL-STD-1750A T1.0-05A** compiler, as described in this Appendix, are provided by **SD-Scicon plc**. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

...

type INTEGER is range -2**15 .. (2**15)-1;
type LONG_INTEGER is range -2**31 .. (2**31)-1;

type FLOAT is digits 6 range -(2**128-2**106) .. (2**128-2**106)
type LONG_FLOAT is digits 9 range -(2**128-2**96) .. (2**128-2**96)

type DURATION is delta 1.0E-4 range -131072.0000 .. 131071.9999;

...

end STANDARD;
```

Appendix F

Implementation-Dependent Characteristics

NOTE

This appendix is not part of the standard definition of the Ada programming language.

This appendix summarizes the following implementation-dependent characteristics of XD Ada:

- Listing the XD Ada pragmas and attributes.
- Giving the specification of the package SYSTEM.
- Presenting the restrictions on representation clauses and unchecked type conversions.
- Giving the conventions for names denoting implementation-dependent components in record representation clauses.
- Giving the interpretation of expressions in address clauses.
- Presenting the implementation-dependent characteristics of the input-output packages.
- Presenting other implementation-dependent characteristics.

F.1 Implementation-Dependent Pragmas

XD Ada provides the following pragmas, which are defined elsewhere in the text. In addition, XD Ada restricts the predefined language pragmas INLINE and INTERFACE, provides pragma VOLATILE in addition to pragma SHARED, and provides pragma SUPPRESS_ALL in addition to pragma SUPPRESS. See Annex B for a descriptive pragma summary.

- CALL_SEQUENCE_FUNCTION (see Annex B)
- CALL_SEQUENCE_PROCEDURE (see Annex B)
- EXPORT_EXCEPTION (see Section 13.9a.3.2)
- EXPORT_FUNCTION (see Section 13.9a.1.2)
- EXPORT_OBJECT (see Section 13.9a.2.2)
- EXPORT_PROCEDURE (see Section 13.9a.1.2)
- IMPORT_EXCEPTION (see Section 13.9a.3.1)
- IMPORT_FUNCTION (see Section 13.9a.1.1)
- IMPORT_OBJECT (see Section 13.9a.2.1)
- IMPORT_PROCEDURE (see Section 13.9a.1.1)
- LEVEL (see Section 13.5.1)
- LINK_OPTION (see Annex B)
- SUPPRESS_ALL (see Section 11.7)
- TITLE (see Annex B)
- VOLATILE (see Section 9.11)

F.2 Implementation-Dependent Attributes

XD Ada provides the following attributes, which are defined elsewhere in the text. See Appendix A for a descriptive attribute summary.

- BIT (see Section 13.7.2)
- MACHINE_SIZE (see Section 13.7.2)
- TYPE_CLASS (see Section 13.7a.2)

F.3 Specification of the Package System

The package SYSTEM for the MIL-STD-1750A is as follows:

F.3.1 Package System for the MIL-STD-1750A Target

```
package SYSTEM is

    type NAME is (MIL_STD_1750A);

    SYSTEM_NAME    : constant NAME := MIL_STD_1750A;
    STORAGE_UNIT   : constant := 16;
    MEMORY_SIZE    : constant := 2**17;
    MIN_INT        : constant := -(2**31);
    MAX_INT        : constant := 2**31-1;
    MAX_DIGITS     : constant := 9;
    MAX_MANTISSA   : constant := 31;
    FINE_DELTA     : constant := 2.0**(-31);
    TICK           : constant := 100.0E-6;
    subtype PRIORITY is INTEGER range 0 .. 15;

    subtype LEVEL is INTEGER range 0 .. 7;

--  Address type
--
    type ADDRESS is private;

    ADDRESS_ZERO : constant ADDRESS;
    type ADDRESS_INT is range -32768 .. 32767;
    function TO_ADDRESS      (X : ADDRESS_INT)                    return ADDRESS;
    function TO_ADDRESS      (X : {universal_integer})            return ADDRESS;
    function TO_ADDRESS_INT  (X : ADDRESS)                        return ADDRESS_INT;

    function "+" (LEFT : ADDRESS;      RIGHT : ADDRESS_INT)  return ADDRESS;
    function "+" (LEFT : ADDRESS_INT;  RIGHT : ADDRESS)      return ADDRESS;
    function "-" (LEFT : ADDRESS;      RIGHT : ADDRESS)      return ADDRESS_INT;
    function "-" (LEFT : ADDRESS;      RIGHT : ADDRESS_INT)  return ADDRESS;

--  function "="  (LEFT, RIGHT : ADDRESS) return BOOLEAN;
--  function "/=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;
    function "<"  (LEFT, RIGHT : ADDRESS) return BOOLEAN;
    function "<=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;
    function ">"  (LEFT, RIGHT : ADDRESS) return BOOLEAN;
    function ">=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;

--
--  Note that because ADDRESS is a private type
--  the functions "=" and "/=" are already available
-- Generic functions used to access memory
--
    generic
        type TARGET is private;
    function FETCH_FROM_ADDRESS (A : ADDRESS) return TARGET;

    generic
        type TARGET is private;
    procedure ASSIGN_TO_ADDRESS (A : ADDRESS; T : TARGET);

    type TYPE_CLASS is (TYPE_CLASS_ENUMERATION,
                        TYPE_CLASS_INTEGER,
                        TYPE_CLASS_FIXED_POINT,
                        TYPE_CLASS_FLOATING_POINT,
                        TYPE_CLASS_ARRAY,
                        TYPE_CLASS_RECORD,
                        TYPE_CLASS_ACCESS,
                        TYPE_CLASS_TASK,
                        TYPE_CLASS_ADDRESS);

--
--  XD Ada hardware-oriented types and functions
--
    type    BIT_ARRAY is array (INTEGER range <>) of BOOLEAN;
    pragma  PACK(BIT_ARRAY);
    subtype BIT_ARRAY_16 is BIT_ARRAY (0 .. 15);
    subtype BIT_ARRAY_32 is BIT_ARRAY (0 .. 31);

    type UNSIGNED_WORD     is range 0 .. 65535;
    for  UNSIGNED_WORD'SIZE     use 16;

    function "not" (LEFT        : UNSIGNED_WORD) return UNSIGNED_WORD;
    function "and" (LEFT, RIGHT : UNSIGNED_WORD) return UNSIGNED_WORD;
    function "or"  (LEFT, RIGHT : UNSIGNED_WORD) return UNSIGNED_WORD;
    function "xor" (LEFT, RIGHT : UNSIGNED_WORD) return UNSIGNED_WORD;

    function TO_UNSIGNED_WORD  (X : BIT_ARRAY_16)   return UNSIGNED_WORD;
    function TO_BIT_ARRAY_16   (X : UNSIGNED_WORD)  return BIT_ARRAY_16;

    type UNSIGNED_WORD_ARRAY is array (INTEGER range <>) of UNSIGNED_WORD;

--
    type UNSIGNED_LONGWORD is range MIN_INT .. MAX_INT;

    for UNSIGNED_LONGWORD'SIZE use 32;

    function "not" (LEFT        : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
    function "and" (LEFT, RIGHT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
    function "or"  (LEFT, RIGHT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
    function "xor" (LEFT, RIGHT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;

    function TO_UNSIGNED_LONGWORD (X : BIT_ARRAY_32)   return UNSIGNED_LONGWORD;
    function TO_BIT_ARRAY_32         (X : UNSIGNED_WORD)  return BIT_ARRAY_32;

    type UNSIGNED_LONGWORD_ARRAY is array (INTEGER range <>) of UNSIGNED_LONGWORD;
```

```
--  Conventional names for static subtypes of type UNSIGNED_LONGWORD
--
    subtype UNSIGNED_1  is UNSIGNED_LONGWORD range 0 .. 2** 1-1;
    subtype UNSIGNED_2  is UNSIGNED_LONGWORD range 0 .. 2** 2-1;
    subtype UNSIGNED_3  is UNSIGNED_LONGWORD range 0 .. 2** 3-1;
    subtype UNSIGNED_4  is UNSIGNED_LONGWORD range 0 .. 2** 4-1;
    subtype UNSIGNED_5  is UNSIGNED_LONGWORD range 0 .. 2** 5-1;
    subtype UNSIGNED_6  is UNSIGNED_LONGWORD range 0 .. 2** 6-1;
    subtype UNSIGNED_7  is UNSIGNED_LONGWORD range 0 .. 2** 7-1;
    subtype UNSIGNED_8  is UNSIGNED_LONGWORD range 0 .. 2** 8-1;
    subtype UNSIGNED_9  is UNSIGNED_LONGWORD range 0 .. 2** 9-1;
    subtype UNSIGNED_10 is UNSIGNED_LONGWORD range 0 .. 2**10-1;
    subtype UNSIGNED_11 is UNSIGNED_LONGWORD range 0 .. 2**11-1;
    subtype UNSIGNED_12 is UNSIGNED_LONGWORD range 0 .. 2**12-1;
    subtype UNSIGNED_13 is UNSIGNED_LONGWORD range 0 .. 2**13-1;
    subtype UNSIGNED_14 is UNSIGNED_LONGWORD range 0 .. 2**14-1;
    subtype UNSIGNED_15 is UNSIGNED_LONGWORD range 0 .. 2**15-1;
    subtype UNSIGNED_16 is UNSIGNED_LONGWORD range 0 .. 2**16-1;
    subtype UNSIGNED_17 is UNSIGNED_LONGWORD range 0 .. 2**17-1;
    subtype UNSIGNED_18 is UNSIGNED_LONGWORD range 0 .. 2**18-1;
    subtype UNSIGNED_19 is UNSIGNED_LONGWORD range 0 .. 2**19-1;
    subtype UNSIGNED_20 is UNSIGNED_LONGWORD range 0 .. 2**20-1;
    subtype UNSIGNED_21 is UNSIGNED_LONGWORD range 0 .. 2**21-1;
    subtype UNSIGNED_22 is UNSIGNED_LONGWORD range 0 .. 2**22-1;
    subtype UNSIGNED_23 is UNSIGNED_LONGWORD range 0 .. 2**23-1;
    subtype UNSIGNED_24 is UNSIGNED_LONGWORD range 0 .. 2**24-1;
    subtype UNSIGNED_25 is UNSIGNED_LONGWORD range 0 .. 2**25-1;
    subtype UNSIGNED_26 is UNSIGNED_LONGWORD range 0 .. 2**26-1;
    subtype UNSIGNED_27 is UNSIGNED_LONGWORD range 0 .. 2**27-1;
    subtype UNSIGNED_28 is UNSIGNED_LONGWORD range 0 .. 2**28-1;
    subtype UNSIGNED_29 is UNSIGNED_LONGWORD range 0 .. 2**29-1;
    subtype UNSIGNED_30 is UNSIGNED_LONGWORD range 0 .. 2**30-1;
    subtype UNSIGNED_31 is UNSIGNED_LONGWORD range 0 .. 2**31-1;

private
    -- Not shown
end SYSTEM;
```

F.4 Restrictions on Representation Clauses

The representation clauses allowed in XD Ada are length, enumeration, record representation, and address clauses.

F.5 Conventions for Implementation-Generated Names Denoting Implementation-Dependent Components in Record Representation Clauses

XD Ada does not allocate implementation-dependent components in records.

F.6 Interpretation of Expressions Appearing in Address Clauses

Expressions appearing in address clauses must be of the type ADDRESS defined in package SYSTEM (see Section 13.7a.1 and Section F.3).

XD Ada allows address clauses for variables (see Section 13.5). For address clauses on variables, the address expression is interpreted as a MIL-STD-1750A 16-bit logical address.

XD Ada supports address clauses on task entries to allow interrupts to cause a reschedule directly. For address clauses on task entries, the address expression is interpreted as a MIL-STD-1750A interrupt number in the range 0 .. 15.

In XD Ada for MIL-STD-1750A, values of type SYSTEM.ADDRESS are interpreted as integers in the range -2^{15} .. 2^{15} -1. As SYSTEM.ADDRESS is a private type, the only operations allowed on objects of this type are those given in package SYSTEM.

F.7 Restrictions on Unchecked Type Conversions

XD Ada supports the generic function UNCHECKED_CONVERSION with the restrictions given in Section 13.10.2.

F.8 Implementation-Dependent Characteristics of Input-Output Packages

The packages SEQUENTIAL_IO and DIRECT_IO are implemented as null packages that conform to the specification given in the *Reference Manual for the Ada Programming Language*. The packages raise the exceptions specified in Chapter 14 of the *Reference Manual for the Ada Programming Language*. The three possible exceptions that are raised by these packages are given here, in the order in which they are raised.

Exception	When Raised
STATUS_ERROR	Raised by an attempt to operate upon or close a file that is not open (no files can be opened).
NAME_ERROR	Raised if a file name is given with a call of CREATE or OPEN.
USE_ERROR	Raised if exception STATUS_ERROR is not raised.

MODE_ERROR cannot be raised since no file can be opened (therefore it cannot have a current mode).

The predefined package LOW_LEVEL_IO is provided.

F.8.1 The Package TEXT_IO

The package TEXT_IO conforms to the specification given in the *Reference Manual for the Ada Programming Language*. String input-output is implemented aѕ defined. File input-output is supported to STANDARD_INPUT and STANDARD_OUTPUT only. The possible exceptions that are raised by package TEXT_IO are as follows:

Exception	When Raised
STATUS_ERROR	Raised by an attempt to operate upon or close a file that is not open (no files can be opened).
NAME_ERROR	Raised if a file name is given with a call of CREATE or OPEN.
MODE_ERROR	Raised by an attempt to read from, or test for the end of, STANDARD_OUTPUT, or to write to STANDARD_INPUT.
END_ERROR	Raised by an attempt to read past the end of STANDARD_INPUT.
USE_ERROR	Raised when an unsupported operation is attempted, that would otherwise be legal.

The type COUNT is defined as follows:

```
type COUNT is range 0 .. INTEGER'LAST;
```

The subtype FIELD is defined as follows:

```
type FIELD is INTEGER range 0 .. 255;
```

F.8.2 The Package IO_EXCEPTIONS

The specification of the package IO_EXCEPTIONS is the same as that given in the *Reference Manual for the Ada Programming Language*.

F.9 Other Implementation Characteristics

Implementation characteristics associated with the definition of a main program, various numeric ranges, and implementation limits are summarized in the following sections.

F.9.1 Definition of a Main Program

Any library procedure can be used as a main program provided that it has no formal parameters.

F.9.2 Values of Integer Attributes

The ranges of values for integer types declared in package STANDARD are as follows:

INTEGER	$-2^{15} .. 2^{15} -1$	(-32768 .. 32767)
LONG_INTEGER	$-2^{31} .. 2^{31} -1$	(-2147483648 .. 2147483647)

For the package TEXT_IO, the range of values for types COUNT and FIELD are as follows:

COUNT	$0 .. 2^{15} -1$	(0 .. 32767)
FIELD	0 .. 255	

F.9.3 Values of Floating-Point Attributes

Floating-point types are described in Section 3.5.7. The representation attributes of floating-point types are summarized in the following table:

	FLOAT	LONG_FLOAT
DIGITS	6	9
SIZE	32	48
MANTISSA	21	31
EMAX	84	124
EPSILON	2^{-20}	2^{-30}
SMALL	2^{-85}	2^{-125}
LARGE	$2^{84}-2^{63}$	$2^{124}-2^{93}$
SAFE_EMAX	127	127
SAFE_SMALL	2^{-126}	2^{-126}
SAFE_LARGE	$2^{127}-2^{106}$	$2^{127}-2^{96}$
FIRST	$-(2^{128}-2^{106})$	$-(2^{128}-2^{96})$
LAST	$2^{128}-2^{106}$	$2^{128}-2^{96}$
MACHINE_RADIX	2	2
MACHINE_MANTISSA	23	39
MACHINE_EMAX	127	127
MACHINE_EMIN	-128	- 128
MACHINE_ROUNDS	FALSE	FALSE
MACHINE_OVERFLOWS	FALSE	FALSE

F.9.4 Attributes of Type DURATION

The values of the significant attributes of type DURATION are as follows:

DURATION'DELTA	1.E-4	(10^{-4})
DURATION'SMALL	2#1.0#E-14	(2^{-14})
DURATION'FIRST	-131072.0000	(-2^{17})
DURATION'LAST	131071.9999	$(2^{17}$-'DELTA)

F.9.5 Implementation Limits

Limit	Description
255	Maximum identifier length (number of characters)
255	Maximum number of characters in a source line
2^{10}	Maximum number of library units and subunits in a compilation closure[1]
2^{12}	Maximum number of library units and subunits in an execution closure[2]
2^{16} -1	Maximum number of enumeration literals in an enumeration type definition
2^{16} -1	Maximum number of lines in a source file
2^{15} x 16	Maximum number of bits in any object
2^{16} -1	Maximum number of exceptions

[1] The compilation closure of a given unit is the total set of units that the given unit depends on, directly and indirectly.

[2] The execution closure of a given unit is the compilation closure plus all associated secondary units.

COMPILE

Forms the closure of one or more specified units. Compiles, from external source files, any unit in the closure (except entered units) that was revised since that unit was last compiled into the current program library. Recompiles, from external copied source files, any unit in the closure that needs to be made current. Completes any incomplete generic instantiations.

Format

COMPILE *unit-name[,...]*

Command Qualifiers	**Defaults**
/AFTER = time	/AFTER = TODAY
/[NO]ANALYSIS_DATA[= file-spec]	/NOANALYSIS_DATA
/BATCH_LOG = file-spec	See text.
/[NO]CHECK	See text.
/CLOSURE	See text
/COMMAND[= file-spec]	See text
/[NO]CONFIRM	/NOCONFIRM
/[NO]COPY_SOURCE	/COPY_SOURCE
/[NO]DEBUG[= (option[,...])]	/DEBUG = ALL
/[NO]DIAGNOSTICS[= file-spec]	/NODIAGNOSTICS
/[NO]ERROR_LIMIT[= n]	/ERROR_LIMIT = 30
/[NO]KEEP	/KEEP
/[NO]LIST[= file-spec]	/NOLIST
/[NO]LOG	/NOLOG
/[NO]MACHINE_CODE	/NOMACHINE_CODE
/NAME = job-name	See text.
/[NO]NOTE_SOURCE	/NOTE_SOURCE
/[NO]NOTIFY	/NOTIFY
/[NO]OPTIMIZE[= (option[,...])]	See text
/OUTPUT = file-spec	/OUTPUT = SYS$OUTPUT
/[NO]PRELOAD	/NOPRELOAD
/[NO]PRINTER[= queue-name]	/NOPRINTER
/QUEUE = queue-name	/QUEUE = XDADA$BATCH
/[NO]SHOW[= option]	/SHOW = PORTABILITY
/SPECIFICATION_ONLY	See text

/SUBMIT	/SUBMIT
/[NO]SYNTAX_ONLY	/NOSYNTAX_ONLY
/WAIT	/SUBMIT
/[NO]WARNINGS[=(option[,...])]	See text.
Positional Qualifiers	**Defaults**
/BODY	See text.
/[NO]DATE_CHECK	/DATE_CHECK
/FORCE_BODY	See text.

Prompt

_Unit:

Command Parameters

unit-name

Specifies one or more units in the current program library the closure of which is to be processed with the COMPILE command. You must express subunit names using selected component notation as follows:

```
ancestor-unit-name.parent-unit-name[...].subunit-name
```

The unit names may include percent signs (%) and asterisks (*) as wildcard characters. Refer to the *VMS DCL Concepts Manual* for more detailed information on wildcard characters.

Description

The XDACS COMPILE command is useful for compiling and recompiling units as you revise the source files of an existing Ada program.

For each set of units specified, the COMPILE command performs the following steps:

1. Forms the closure of the specified units.
2. Looks up the source file for each unit in the closure that has been compiled or copied (not entered) into the current program library. Unless otherwise specified with the XDACS SET SOURCE command, the source-file-directory search order is as follows:
   a. SYS$DISK:[] (the current default directory)

   b. ;0 (the directory that contained the file when it was last compiled), or node::;0 (if the file specification of the source file being compiled contains a node name)

   The search order takes precedence over the version number or revision date-time if different versions of a source file exist in two or more directories. Within any one directory, the version of a particular file that has the highest number is considered for compilation.

3. Compares the revision date-time of each source file with that of the version last noted in the program library by the /NOTE_SOURCE compiler qualifier (the qualifier is used with the COMPILE, RECOMPILE and DCL XDADA commands).

4. Notes for compilation any source file with a revision date-time later than that noted in the program library.

5. Notes any unit in the closure that had to be, or will have to be, recompiled to make all units in the closure current.

   Note that the compiler recompiles obsolete units from copied source files (.ADC). If a needed copied source file is missing, the file is identified and no recompilations are done. Copied source files are created when the /COPY_SOURCE qualifier is in effect (the default for the COMPILE, RECOMPILE and DCL XDADA commands).

   If the closure you are recompiling includes an obsolete entered unit, that unit is not affected by the COMPILE command; an error diagnostic is issued and the COMPILE command is not executed. You should recompile an obsolete entered unit in its own program library and then reenter it into the current program library before you try to recompile its dependent units in the current library.

6. Creates a DCL command file for the compiler. The file contains commands to compile the appropriate units from source files and to recompile any obsolete units from copied source files, in the proper order. Entered units are not considered for compilation or recompilation. The command file is deleted after the COMPILE command is terminated, unless you specified the /COMMAND qualifier. If you specified the /COMMAND qualifier, the command file is retained for future use, and the compiler is not invoked.

7. If you did not specify the /COMMAND qualifier, the appropriate XD Ada compiler is invoked as follows:

   a. By default (COMPILE/SUBMIT), the compiler command file generated in step 6 is submitted as a batch job.

b. If you specified the /WAIT qualifier, the command file is executed in a subprocess. You must wait for the compilation to terminate before issuing another command. Note that when you specify the COMPILE/WAIT command, process logical names are propagated to the subprocess generated to execute the command file.

XDACS output originating before the compiler is invoked is reported to your terminal by default, or to a file specified with the /OUTPUT qualifier. Compiler diagnostics are reported to a log file by default or to the terminal if the COMPILE command is executed in a subprocess (COMPILE/WAIT).

See Chapter 3 for more information on the COMPILE command.

Command Qualifiers

/AFTER=time

Requests that the batch job be held until after a specific time when the COMPILE command is executed in batch mode (the default mode). If the specified time has already passed, or if the /AFTER qualifier is not specified, the job is queued for immediate processing.

You can specify either an absolute time or a combination of absolute and delta time. See the *VMS DCL Concepts Manual* (or use HELP Specify Date_Time at the DCL prompt) for complete information on specifying time values.

/ANALYSIS_DATA[=file-spec]
/NOANALYSIS_DATA (D)

Controls whether a data analysis file containing source code cross-reference and static analysis information is created. The data analysis file is supported only for use with DIGITAL layered products, such as the VAX Source Code Analyzer.

One data analysis file is created for each source file compiled and for each copied unit that is recompiled. The default directory for data analysis files is the current default directory. The default file name is the name of the source file being compiled. The default file type is .ANA. No wildcard characters are allowed in the file specification.

By default, no data analysis file is created.

COMPILE

/BATCH_LOG=file-spec

Provides a file specification for the batch log file when the COMPILE command is executed in batch mode (the default mode).

If you do not give a directory specification with the *file-spec* option, the batch log file is created by default in the current default directory. If you do not give a file specification, the default file name is the job name specified with the /NAME=job-name qualifier. If no job name has been specified, the program library manager creates a file name comprising up to the first 39 characters of the first unit name specified. If no job name has been specified and there is a wildcard character in the first unit specified, the program library manager uses the default file name XDACS_COMPILE. The default file type is .LOG. No wildcard characters are allowed in the file specification.

/CHECK
/NOCHECK

Controls whether all run-time checks are suppressed. The /NOCHECK qualifier is equivalent to having all possible SUPPRESS pragmas in the source code.

Explicit use of the /CHECK qualifier overrides any occurrences of the pragmas SUPPRESS and SUPPRESS_ALL in the source code, without the need to edit the source code.

By default, run-time checks are only suppressed in cases where a pragma SUPPRESS or SUPPRESS_ALL appears in the source code.

See the *XD Ada MIL-STD-1750A Supplement to the Ada Language Reference Manual* for more information on the pragmas SUPPRESS and SUPPRESS_ALL.

/CLOSURE

Forces the compilation of all units in the closure of the set of units named in the COMPILE command; can be used only with the /NODATE_CHECK qualifier. See the description of the /[NO]DATE_CHECK qualifier in the list of positional qualifiers.

/COMMAND[=file-spec]

Controls whether the compiler is invoked as a result of the COMPILE command, and determines whether the command file generated to invoke the compiler is saved. If you specify the /COMMAND qualifier, XDACS does not invoke the compiler, and the generated command file is saved for you to invoke or submit as a batch job.

The *file-spec* option allows you to enter a file specification for the generated command file. The default directory for the command file is the current default directory. By default, XDACS provides a file name comprising up to the first 39 characters of the first unit name specified If there is a wildcard character in the first unit specified, the compiler uses the default file name XDACS COMPILE. The default file type is .COM. No wildcard characters are allowed in the file specification.

By default, if you do not specify the *file-spec* option, the program library manager deletes the generated command file when the COMPILE command completes normally or is terminated.

/CONFIRM
/NOCONFIRM (D)

Controls whether the COMPILE command asks you for confirmation before performing a possibly lengthy operation. If you specify the /CONFIRM qualifier, the possible responses are as follows:

- Affirmative responses are YES, TRUE, and 1
- Negative responses are NO, FALSE, 0, and the RETURN key

You can use any combination of upper and lowercase letters for word responses. Word responses can be abbreviated to one or more letters (for example, Y, YE, or YES). If you type a response other than one of those in the list, the prompt is reissued.

By default, no confirmation is requested.

/COPY_SOURCE (D)
/NOCOPY_SOURCE

Controls whether a copied source file (.ADC) is created in the current program library when a compilation unit is compiled without error. Recompilation requires that a copied source file exist in the current program library for any unit that is to be recompiled.

By default, a copied source file is created in the current program library when a unit is compiled without error.

/DEBUG[=(option[,...])] (D)
/NODEBUG

Controls which debugger compiler options are provided. You can debug XD Ada programs with the XD Ada Debugger (see Chapter 11). You can request the following options:

ALL	Provides both SYMBOLS and TRACEBACK
NONE	Provides *neither* SYMBOLS nor TRACEBACK
[NO]SYMBOLS	Controls whether a debugger symbol table is created for the object file
[NO]TRACEBACK	Controls whether traceback (a subset of the debugger symbol information) information is included in the object file

By default, both debugger symbol records and traceback information are included in the object files (/DEBUG = ALL, or equivalently: /DEBUG).

/DIAGNOSTICS[= file-spec]
/NODIAGNOSTICS (D)

Controls whether a diagnostics file containing compiler messages and diagnostic information is created. The diagnostics file is supported only for use with DIGITAL layered products, such as the VAX Language Sensitive Editor.

A diagnostics file is created for each source file compiled. The default directory for diagnostics files is the current default directory. The default file name is the name of the source file being compiled. The default file type of a diagnostics file is .DIA. No wildcard characters are allowed in the file specification.

By default, no diagnostics file is created.

/ERROR_LIMIT[= n]
/NOERROR_LIMIT

Controls whether execution of the COMPILE command for a given compilation unit is terminated upon the occurrence of the nth E-level error within that unit.

Error counts are not accumulated across a sequence of compilation units. If the /ERROR_LIMIT = n option is specified, each compilation unit may have up to n - 1 errors without terminating the compilation. When the error limit is reached within a compilation unit, compilation of that unit is terminated, but compilation of subsequent units continues.

The /ERROR_LIMIT = 0 qualifier is equivalent to ERROR_LIMIT = 1.

By default, execution of the COMPILE command is terminated for a given compilation unit upon the occurrence of the 30th E-level error within that unit (equivalent to /ERROR_LIMIT = 30).

COMPILE

/KEEP (D)
/NOKEEP

Controls whether the batch log file generated is deleted after it is printed when the COMPILE command is executed in batch mode (the default mode).

By default, the log file is not deleted.

/LIST[=file-spec]
/NOLIST (D)

Controls whether a listing file is created. One listing file is created for each compilation unit (not file) compiled or recompiled by the COMPILE command.

The default directory for listing files is the current default directory. The default file name of a listing file corresponds to the name of its compilation unit and uses the XD Ada file-name conventions described in Appendix C. The default file type of a listing file is .LIS. No wildcard characters are allowed in the file specification.

By default, the COMPILE command does not create a listing file.

/LOG
/NOLOG (D)

Controls whether a list of all the ~~units that mu~~st be compiled or recompiled is displayed.

By default, a list of the units that must be compiled or recompiled is not displayed.

/MACHINE_CODE
/NOMACHINE_CODE (D)

Controls whether generated machine code (approximating assembly language notation) is included in the listing file.

By default, generated machine code is not included in the listing file.

/NAME=job-name

Specifies a string to be used as the job name and as the file name for the batch log file when the COMPILE command is executed in batch mode (the default mode). The job name can have from 1 to 39 characters.

COMPILE

By default, if you do not specify the /NAME qualifier, the program library manager creates a job name comprising up to the first 39 characters of the first unit name specified. If you do not specify the /NAME qualifier, but use a wildcard character in the first unit name specified, the compiler uses the default name XDACS_COMPILE. In these cases the job name is also the file name of the batch log file.

/NOTE_SOURCE (D)
/NONOTE_SOURCE

Controls whether the file specification of the source file is noted in the program library when a unit is compiled without error. The COMPILE command uses this information to locate revised source files.

By default, the file specification of the source file is noted in the current program library when a unit is compiled without error.

/NOTIFY (D)
/NONOTIFY

Controls whether a message is broadcast when the COMPILE command is executed in batch mode (the default mode). The message is broadcast to any terminal at which you are logged in, notifying you that your job has been completed or terminated.

By default, a message is broadcast.

/OPTIMIZE[=(option[,...])]
/NOOPTIMIZE

Controls the level of optimization that is applied in producing the compiled code. You can specify one of the following primary options:

TIME

Provides full optimization with time as the primary optimization criterion. Overrides any occurrences of the pragma OPTIMIZE(SPACE) in the source code.

SPACE

Provides full optimization with space as the primary optimization criterion. Overrides any occurrences of the pragma OPTIMIZE(TIME) in the source code.

DEVELOPMENT
Suggested when active development of a program is in progress. Provides some optimization, but development considerations and ease of debugging take preference over optimization. This option overrides pragmas that establish a dependence on a subprogram (the pragma INLINE), and thus reduces the need for recompilations when such bodies are modified.

NONE
Provides no optimization. Suppresses expansions in line of subprograms, including those specified by the pragma INLINE.

The /NOOPTIMIZE qualifier is equivalent to /OPTIMIZE=NONE.

By default, the COMPILE command applies full optimization with space as the primary optimization criterion (like /OPTIMIZE=SPACE, but observing uses of the pragma OPTIMIZE).

The /OPTIMIZE qualifier also has a set of secondary options that you can use separately or together with the primary options to override the default behavior for expansion in line.

The INLINE secondary option can have the following values (see the *XD Ada MIL-STD-1750A Run-Time Reference Manual* for more information about expansion in line):

INLINE:NONE
Disables subprogram expansion in line. This option overrides any occurrences of the pragma INLINE in the source code, without your having to edit the source file. It also disables implicit expansion in line of subprograms. ("Implicit expansion in line" means that the compiler assumes a pragma INLINE for certain subprograms as an optimization.) A call to a subprogram in another unit is not expanded in line, regardless of other /OPTIMIZE options in effect when that unit was compiled.

INLINE:NORMAL
Provides normal subprogram expansion in line.

Subprograms to which an explicit pragma INLINE applies are expanded in line under certain conditions. In addition, some subprograms are implicitly expanded in line. The compiler assumes a pragma INLINE for calls to some small local subprograms (subprograms that are declared in the same unit as the unit in which the call occurs).

INLINE:SUBPROGRAMS Provides maximal subprogram expansion in line

In addition to the normal subprogram expansion in line that occurs when INLINE:NORMAL is specified this option results in implicit expansion in line of some small subprograms declared in other units. The compiler assumes a pragma INLINE for any subprogram it it improves execution speed and reduces code size. This option may establish a dependence on the body of another unit, as would be the case if a pragma INLINE were specified explicitly in the source code.

INLINE:MAXIMAL Provides maximal subprogram expansion in line.

Maximal subprogram expansion in line occurs as for *INLINE*:SUBPROGRAMS.

By default, the /OPTIMIZE qualifier primary options have the following secondary-option values:

OPTIMIZE = TIME	=(INLINE:NORMAL)
OPTIMIZE = SPACE	=(INLINE:NORMAL)
OPTIMIZE = DEVELOPMENT	=(INLINE:NONE)
OPTIMIZE = NONE	=(INLINE:NONE)

See Chapter 3 of Version 2.0 of *Developing Ada Programs on VMS Systems* for more information on the /OPTIMIZE qualifier and its options.

/OUTPUT = file-spec

Requests that any XDACS output generated before the compiler is invoked be written to the file specified rather than to SYS$OUTPUT. Any diagnostic messages are written to both SYS$OUTPUT and the file.

The default directory is the current default directory. If you specify a file type but omit the file name, the default file name is XDACS. The default file type is .LIS. No wildcard characters are allowed in the file specification.

By default, the COMPILE command output is written to SYS$OUTPUT.

/PRELOAD
/NOPRELOAD (D)

Controls whether the COMPILE command processes external source files so that new compilation units or unit dependences introduced in those files, or any new source files previously processed by the DCL XDADA command, are accounted for. Preload processing involves the partial compilation and syntax checking of the following files:

- Any external source files with a creation date-time later than that noted in the program library
- Any new units introduced into the closure of units specified by way of modifications to the known external source files (preload processing does not include new external source files that are not already accounted for in the program library)

Preload processing is done immediately, after the creation date-time of each external source file is checked, and before the usual COMPILE compilations and recompilations are performed. If you have also specified the /CONFIRM qualifier, you are prompted for confirmation for each external file to be preloaded.

By default, the COMPILE command does not process external source files to account for new compilation units or unit dependences.

/PRINTER[=queue-name]
/NOPRINTER (D)

Controls whether the batch job log file is queued for printing when the COMPILE command is executed in batch mode (the default mode).

The /PRINTER qualifier allows you to specify a particular print queue. The default print queue for the log file is SYS$PRINT.

By default, the log file is not queued for printing. If you specify the /NOPRINTER qualifier, the /KEEP qualifier is assumed.

/QUEUE=queue-name

Specifies the batch job queue in which the job is entered when the COMPILE command is executed in batch mode (the default mode).

By default, if the /QUEUE qualifier is not specified, XDACS first checks whether the logical name XDADA$BATCH is defined. If it is, XDACS enters the job in the queue specified. Otherwise the job is placed in the default system batch job queue, SYS$BATCH.

COMPILE

/SHOW[=option] (D)
/NOSHOW

Controls the listing file options included when a listing file is provided. You can specify one of the following options:

ALL	Provides all listing file options.
[NO]PORTABILITY	Controls whether a program portability summary is included in the listing file. By default, the COMPILE command provides a portability summary (/SHOW=PORTABILITY). See Appendix E for details of what can appear in a portability summary. See Chapter 5 of Version 2.0 of *Developing Ada Programs on VMS Systems* for more information on program portability.
NONE	Provides none of the listing file options (same as /NOSHOW).

/SPECIFICATION_ONLY

Causes only the specifications of the units specified to be considered for compilation. You can use the /CLOSURE qualifier with the /SPECIFICATION_ONLY qualifier to force only the specifications in the execution closure of the specified units to be considered for compilation.

By default, if the /SPECIFICATION_ONLY qualifier is omitted, all of the specifications, bodies, and subunits in the execution closure of the units specified are considered for compilation.

/SUBMIT

Directs XDACS to submit the command file generated for the compiler to a batch queue. You can continue to issue commands in your current process without waiting for the batch job to complete. The compiler output is written to a log file.

By default, XDACS submits the command file generated for the compiler to a batch queue.

/SYNTAX_ONLY
/NOSYNTAX_ONLY (D)

Controls whether the source file is to be checked only for correct syntax. If you specify the /SYNTAX_ONLY qualifier, other compiler checks are not performed (for example, semantic analysis, type checking, and so on), and the program library is not updated.

By default, the compiler performs all checks.

/WAIT

Directs XDACS to execute the command file generated for the compiler in a subprocess. Execution of your current process is suspended until the subprocess completes. The compiler output is written directly to your terminal. Note that process logical names are propagated to the subprocess generated to execute the command file.

By default, XDACS submits the command file generated for the compiler to a batch queue (COMPILE/SUBMIT).

/WARNINGS[=(message-option[,...])]
/NOWARNINGS

Controls which categories of informational (I-level) and warning (W-level) messages are displayed and where those messages are displayed. You can specify any combination of the following message options:

WARNINGS: (*destination*[,...])
NOWARNINGS

WEAK_WARNINGS: (*destination*[,...])
NOWEAK_WARNINGS

SUPPLEMENTAL: (*destination*[,...])
NOSUPPLEMENTAL

COMPILATION_NOTES: (*destination*[,...])
NOCOMPILATION_NOTES

STATUS: (*destination*[,...])
NOSTATUS

The possible values of *destination* are ALL, NONE, or any combination of TERMINAL (terminal device), LISTING (listing file), and DIAGNOSTICS (diagnostics file). The message categories are summarized as follows (see Chapter 3 of the Version 2.0 of *Developing Ada Programs on VMS Systems* for more information):

COMPILE

WARNINGS — W-level: Indicates a definite problem in a legal program, for example, an unknown pragma.

WEAK WARNINGS — I-level: Indicates a potential problem in a legal program, for example, a possible CONSTRAINT_ERROR at run-time. These are the only kind of I-level messages that are counted in the summary statistics at the end of a compilation.

SUPPLEMENTAL — I-level: Additional information associated with preceding E-level or W-level diagnostics.

COMPILATION NOTES — I-level: Information about how the compiler translated a program, such as record layout, parameter-passing mechanisms, or decisions made for the pragmas INLINE, INTERFACE, or the import-subprogram pragmas.

STATUS — I-level: End of compilation statistics and other messages.

The defaults are as follows:

```
/WARNINGS=(WARN:ALL,WEAK:ALL,SUPP:ALL,COMP:NONE,STAT:NONE)
```

If you specify only some of the message categories with the /WARNINGS qualifier, the default values for the other categories are used.

Positional Qualifiers

/BODY

Forces the compilation of the body and subunits (if any) of a specified compilation unit, without forcing the compilation of the specification. You can use the /BODY qualifier only with the /NODATE_CHECK qualifier.

By default, if you use the /NODATE_CHECK qualifier without the /BODY qualifier, the COMPILE command forces the compilation of the specification, as well as the body and any subunits.

/DATE_CHECK (D)
/NODATE_CHECK

Controls whether the COMPILE command checks the creation date and time of source files to determine whether any source files have been revised but not compiled into the current program library. If you specify the /NODATE_CHECK qualifier, the COMPILE command forces the compilation of every unit specified, even though the source file has not been revised since the unit was last compiled; bodies and subunits of the specified units are also recompiled as necessary, to make them current. Entered units are not considered for compilation or recompilation when the /NODATE_CHECK qualifier is in effect.

If you specify the /NODATE_CHECK/CLOSURE qualifier, the COMPILE command forces the compilation of every unit in the closure of the units specified.

If you specify the /NODATE_CHECK/BODY qualifier, the COMPILE command forces the compilation of the body and subunits (if any) of a compilation unit, without forcing the compilation of the specification.

You can use the /NODATE_CHECK qualifier to force the compilation of a set of units using a particular combination of compiler qualifiers.

By default, the COMPILE command checks the creation date and time of source files (/DATE_CHECK) and compiles only the source files that were revised but not compiled into the current program library.

/FORCE_BODY

Forces the compilation or recompilation of the bodies of the specified compilation units, regardless of whether or not the external source files have been modified or the bodies are obsolete.

The /FORCE_BODY qualifier can have different effects depending on its position in the command line, and its interaction with other qualifiers:

- If you append the /FORCE_BODY qualifier to the COMPILE command string (as opposed to appending it to an individual unit parameter), the COMPILE command forces the compilation of the bodies of each unit specified on the command line.
- If you append the /FORCE_BODY qualifier to an individual unit parameter, the COMPILE command forces the compilation of the body of only that unit.
- If you specify the /FORCE_BODY qualifier with the /CLOSURE qualifier, the COMPILE command forces the compilation of the bodies of all the units in the execution closure of the units specified.

By default, if the /FORCE_BODY qualifier is omitted, the specifications, bodies, and subunits of all the units in the execution closure of the units specified are considered for compilation or recompilation.

COMPILE

Examples

1. XDA [illegible]

```
%XDACS-I-CL_COMPILE1, The following units will be compiled [illegible]
        [illegible]
MODEL_INTERFACE
    package specification                    -May-  [illegible]
        USER:[DEMO.CONTROL_LOOP]MODEL_INTERFACE_.ADA
    package body                             -May-  [illegible]
        USER:[DEMO.CONTROL_LOOP]MODEL_INTERFACE.ADA

%XDACS-I-CL_RECOMPILE1, The following units will be recompiled
MOVE_TUBE
    procedure body                           -May-  [illegible]
POSITION_OF_TUBE
    function body                            -May-  [illegible]

%XDACS-I-CL_SUBMITTED, Job MODEL_INTERFACE (queue ALL_BATCH, entry [illegible])
    started on FAST_BATCH
```

The COMPILE command with the /LOG qualifier lists all units in the closure of unit MODEL_INTEPFACE that need to be compiled and recompiled, then submits the compiler command file generated by XDACS as a batch job.

2. XDACS> [illegible]

This command forces the compilation (/NODATE_CHECK) of the entire closure (/CLOSURE) of unit CONTROL_LOOP with the /OPTIMIZE=SPACE qualifier.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below:

Name and Meaning	Value
$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	16
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..254=>’A’, 255=>1)
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..254=>’A’, 255=>2)
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..127=>’A’, 128=>3, 129..255=>’A’)
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..127=>’A’, 128=>4, 129..255=>’A’)
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length	(1..252=>0, 253..255=>298)
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..249=>0, 250..255=>69.0E1)
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	(1..127=>’A’)

$BIG_STRING2 (1..127=>'A', 128=>1)

A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.

$BLANKS (1..235=>' ')

A sequence of blanks twenty characters less than the size of the maximum line length.

$COUNT_LAST 32767

A universal integer literal whose value is TEXT_IO.COUNT'LAST.

$DEFAULT_MEM_SIZE 131072

An integer literal whose value is SYSTEM.MEMORY_SIZE.

$DEFAULT_STOR_UNIT 16

An integer literal whose value is SYSTEM.STORAGE_UNIT.

$DEFAULT_SYS_NAME MIL_STD_1750A

The value of the constant SYSTEM.SYSTEM_NAME.

$DELTA_DOC 2.0**(-31)

A real literal whose value is SYSTEM.FINE_DELTA.

$FIELD_LAST 255

A universal integer literal whose value is TEXT_IO.FIELD'LAST.

$FIXED_NAME NO_SUCH_TYPE

The name of a predefined fixed-point type other than DURATION.

$FLOAT_NAME NO_SUCH_TYPE

The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.

$GREATER_THAN_DURATION 75000.0

A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.

$GREATER_THAN_DURATION_BASE_LAST 131073.0

A universal real literal that is greater than DURATION'BASE'LAST.

$HIGH_PRIORITY 15

An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.

$ILLEGAL_EXTERNAL_FILE_NAME1 BADCHAR@.!

An external file name which contains invalid characters.

$ILLEGAL_EXTERNAL_FILE_NAME2 THIS_IS_A_FILE_NAME_WITH_MORE_THAN_70_CHARACTERS_IN_IT_TO_ALLOW_THE_MAC

An external file name which is too long.

$INTEGER_FIRST -32768

A universal integer literal whose value is INTEGER'FIRST.

$INTEGER_LAST 32767

A universal integer literal whose value is INTEGER'LAST.

$INTEGER_LAST_PLUS_1 32768

A universal integer literal whose value is INTEGER'LAST+1.

$LESS_THAN_DURATION -75000.0

A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.

$LESS_THAN_DURATION_BASE_FIRST -131073.0

A universal real literal that is less than DURATION'BASE'FIRST.

$LOW_PRIORITY 0

An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.

$MANTISSA_DOC 31

An integer literal whose value is SYSTEM.MAX_MANTISSA.

$MAX_DIGITS 9

Maximum digits supported for floating-point types.

$MAX_IN_LEN 255

Maximum input line length permitted by the implementation.

$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..2=>’2:’, 3..252=>’0’, 253..255=>’11:’)
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..3=>’16:’ 4..251=>’0’, 252..255=>’F.E:’)
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	(1=>’"’, 2..254=>’A’, 255=>’"’)
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	64
$NAME A name ~~of a~~ predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	NO_SUCH_TYPE_AVAILABLE
$NAME_LIST A list of enumeration literals in the type SYSTEM.NAME, separated by commas.	MIL_STD_1750A
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#

$NEW_MEM_SIZE 131072

An integer literal whose value is a permitted argument for pragma memory_size, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE.

$NEW_STOR_UNIT 16

An integer literal whose value is a permitted argument for pragma storage_unit, other than $DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.

$NEW_SYS_NAME MIL_STD_1750A

A value of the type SYSTEM.NAME, other than $DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.

$TASK_SIZE 16

An integer literal whose value is the number of bits required to hold a task object which has a single entry with one inout parameter.

$TICK 0.0001

A real literal whose value is SYSTEM.TICK.

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

E28005C This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

A39005G This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E This test contains an unitended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

C97116A This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

BC3009B This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]
These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84N & M, & CD5011O [5 tests]
These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C
These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B This test wrongly expects an implicitly declared subprogram to be at the the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]
These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK--particular instances of change may be less (line 29).

CD7203B, & CD7204B
These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.